Appendix A

The Dtools Package

This appendix contains `Dtools` package code, as well the package documentation. The package was written for *Maple V* Releases 2 and 3. The programs were run on a Hewlett-Packard 9000/735 workstation. The computer time required to run the programs was minimal, essentially instantaneous.

The `Dtools` were written specifically for the stationary, axially symmetric, perfect fluid field equations derived in Chapter Two. The `Dtools` were thus designed for this narrow application, limiting their applicability, and they require significant operator interaction. Before `Dtools` can be invoked the following conditions must be met:

(i) `Dlie` requires the `liesymm` package,

(ii) A globally defined, closed set of forms of equal rank must be defined in a list R,

(iii) The forms must be annulled with `liesymm's annul` command.

The forms must be of equal rank because `Dtools` assumes that the λ multipliers in (3.8) are constants, which is only true if all forms are of equal rank. When these conditions are met, the `Dtools` routines can be safely used.

The `Dtools` routines are five main procedures: `Dlie, Dlie2, Dsub, Dsimp` and `Dadd`, and three "hidden routines" called by the main procedures `Dreduce, Dclean,` and `Dclean2`. In practice the code for these programs was contained in a file `Dtools.m`. A short program `grforms` loaded the `liesymm` package, `Dtools.m,` defined the forms, and called the `annul` command of `liesymm` . The main routines are discussed below with notes for their implementation and an example of their use. The code for `grforms` and all `Dtools` routines is included at the end of this appendix.

Procedure Dlie

Calling sequence:

```
Dlie(form,multiplier name,ES name,ZS name);
```

Parameters

```
form                --    a single form from the set R
multiplier name     --    root name for multipliers
                    --    created by the procedure
ES name             --    name for the ES equation list
ZS name             --    name for the ZS equation list
```

Global Variables Created

`Dlie` is not a procedure according to the strict *Maple* definition that a procedure returns only a single value. In order to return several variables, `Dlie` creates corresponding global variables. These global variables are:

```
mm        --    a table of equations with multipliers
om        --    a table of equations without multipliers
mmnum     --    the number of elements in mm
omnum     --    the number of elements in om
V         --    an isovector name
backmm    --    a second copy of mm
backom    --    a second copy of om
```

Explanations

- This procedure employs the Harrison-Estabrook technique to produce differential equations for the isovector components obtained from

$$\pounds_V \omega^\alpha = \lambda_\beta \omega^\beta. \tag{A.1}$$

  The single form ω^α is passed to the procedure in the first argument `form`. The procedure first expands the left-hand side of (A.1) by calling the `Lie` command in the `liesymm` package to perform the Lie derivative and section the resulting form.
- The procedure then expands the right-hand side, using multiplier names derived from the root `multiplier name`, the second argument. For example, if the root name is `c`, the multipliers are labeled `c1, c2, c3,` etc. Like 2-forms are collected, and their coefficients set to zero to obtain the isovector component differential equations.
- `Dlie` uses the `Dsimp` routine to simplify the equations as much as possible before returning them. `Dlie` calls `Dsimp` with a third "dummy"

argument, prompting `Dsimp` to call `Dclean2` to search for any multipliers that may be zero. (See the discussion of `Dsimp` below.)

- After no further simplifications can be made, the equations are sorted into three categories: equations with multipliers, multiple-term equations, and single-term, or "trivial" equations. (See the discussion in section 3.3)
  - The equations with multipliers are returned in the table `mm`.
  - The multiple-term equations are returned in the list `esname`.
  - The trivial equations are returned in the set `zsname`.
- Finally, `Dlie` prints all three types of equations, as well as information regarding the number of equations eliminated and moved during the simplification process.

Procedure Dlie2

Calling sequence:

```
Dlie(derivative,multiplier name,ES name,ZS name);
```

Parameters

```
derivative          --     the Lie derivative of a form from R
multiplier name     --     root name for multipliers
ES name             --     name for the ES equation list
ZS name             --     name for the ZS equation list
```

Explanations

- The procedure is almost identical to `Dlie`, except that it assumes that the Lie derivative has already been performed. The procedure expands the right-hand side of (A.1) and equates it with `derivative`. After this point the procedure is identical to `Dlie`.
- This procedure is useful when the Lie derivative from `Lie` must be manipulated by hand. This procedure was used to correct the Lie derivatives for forms containing p and μ, which `liesymm` considered constants. See the discussion in Section 3.3.

Procedure Dsub

Calling sequence:

```
Dsub(equation, multiplier, target equations);
```

Parameters

```
equation           --    an equation containing a multiplier
multiplier         --    the multiplier in equation
target equations   --    equation list
```

Explanation

- This procedure helps eliminate the multipliers from the equations in table `mm`.
- The procedure solves the equation in the first argument for the multiplier specified in the second argument. The resulting expression is substituted into each equation contained in the third argument.
- The third argument contains a number or numbers referring to equations in `mm`. The argument may be a single number, or a list or set of numbers.

Procedure Dsimp

Calling sequence:

```
Dsimp(ES, ZS, dummy);
```

Parameters

```
ES name    --    list of ES equations to simplify
ZS name    --    set of ZS equations to make simplifications
dummy      --    an optional argument acting as a flag.
```

Explanations

- This procedure uses a set of trivial equations (ZS list) to simplify equations in ES list.
- The third argument acts as a flag. When `Dsimp` is called with three arguments, the procedure assumes that the equations still contain multipliers, which the procedure attempts to eliminate in simple cases. When called with two arguments, `Dsimp` assumes that ES contains no multipliers. `Dlie` calls `Dsimp` with three arguments, but generally only two are used.

- `Dsimp` first calls the `Dreduce` procedure. This procedure searches all ES equations for terms contained in ZS. When these terms are found, they are set equal to zero.
- When `Dsimp` is called with two arguments, `Dsimp` then calls the procedure `Dclean`. `Dclean` searches ES for trivial equations created by the simplification process. If these are found, they are eliminated from ES and transferred to ZS.
- When `Dlie` calls `Dsimp` with three arguments, `Dsimp` calls `Dreduce` as before, but then calls `Dclean2` instead of `Dclean`. `Dclean2` performs the same function as `Dclean`, but additionally looks for any multipliers that equal zero. When these multipliers are found, they are set equal to zero in all equations of ES.
- `Dsimp` calls the `Dclean` and `Dclean2` procedures with three arguments, one of which is a logic variable. If `Dclean` makes any changes, the logic variable is set equal to `true`, otherwise the variable is set to `false`. `Dsimp` tests the logic variable, and if `true` the entire process is repeated again, and if `false` the routine ends.

Procedure Dadd

Calling sequence:

```
Dadd(ES, number);
```

Parameters

```
ES        --    list of non-trivial equations
number    --    numbers of equations in mm to add to ES
```

Explanations

- After multipliers have been eliminated from the table `mm`, they can be added to added to ES with `Dadd`. `Equation number` can be a single number, or a set or list of numbers.
- As an example, if the multipliers in equations `mm[3]`, `mm[5]` and `mm[11]` have been eliminated using `Dsub`, these equations can be added to the non-trivial equation list `es1` with the command `Dadd(es1,[3,5,11])`.

Appendix A: The Dtools Package

The grforms Program

The `grforms` program initializes the system's environment. The program code follows this short explanation of its functions. The `grforms` program:

- loads the `liesymm` and `Dtools` packages,
- defines the dependent and independent variables with the `setup` command,
- defines the forms, and places them in a list R,
- annuls the forms with the `annul` command,
- creates the isovector with a component for every independent and dependent variable. Then creates an alias for each component to simplify the output,
- creates simple expressions for dp/df and $d\mu/df$.

```
with(liesymm):
read(`Dtools.m`):
setup(f,y,s,a,b,c,k,h,w,g,r,z):

xi1 := -d(y)&^d(r) + b*(d(z)&^d(r)): # R[1]
xi2 := -d(y)&^d(z) + a*(d(r)&^d(z)): # R[2]
eta1 := -d(f)&^d(r) + k*(d(z)&^d(r)): # R[3]
eta2 := -d(f)&^d(z) + c*(d(r)&^d(z)): # R[4]
zeta1 := -d(s)&^d(r) + w*(d(z)&^d(r)): # R[5]
zeta2 := -d(s)&^d(z) + h*(d(r)&^d(z)): # R[6]

f1 := d(a)&^d(r) + d(b)&^d(z): # R[7]
f2 := d(c)&^d(r) + d(k)&^d(z): # R[8]
f3 := d(h)&^d(r) + d(w)&^d(z): # R[9]

T1 := d(a)&^d(z) - d(b)&^d(r) + ((1/s)*(h*a + w*b) - \
  (2/f)*(a*c + b*k))*(d(r)&^d(z)): # R[10]
T2 := d(c)&^d(z) - d(k)&^d(r) + ((1/f)*(-c^2 - k^2 + a^2 + b^2) + \
  (1/s)*(h*c + w*k) - 2*exp(2*g)*(m + 3*p))*(d(r)&^d(z)): # R[11]
T3 := w*(d(g)&^d(z)) - h*(d(g)&^d(r)) + d(h)&^d(r) - \
  (s/(2*f^2))*(c*k + a*b)*(d(r)&^d(z)): # R[12]
T4 := 2*h*(d(g)&^d(z)) + 2*w*(d(g)&^d(r)) - d(h)&^d(z) - d(w)&^d(r) \
  - (s/(2*f^2))*(a^2 + c^2 -b^2 - k^2)*(d(r)&^d(z)): # R[13]
T5 := d(h)&^d(z) - d(w)&^d(r) - (4*s*p/f)*exp(2*g)*(d(r)&^d(z)): # R[14]

R := [xi1,xi2,eta1,eta2,zeta1,zeta2,f1,f2,f3,T1,T2,T3,T4,T5]:
annul(R,[r,z]):

vars := wedgeset(0):
varlist := [vars]:
numvars := nops(varlist):
```

```
[V.(1..numvars)]:
for i from 1 to numvars do
 alias(V[varlist[i]] = (V[i])(vars))
od:

pf := -(1/(2*f))*(m + p):
mf := A(f):
alias(phi=phi(r,z)):
```

Dtools Code

The complete code for the `Dtools` routines concludes this appendix.

```
Dlie := proc(F,cc,esname,zsname)
#
# Version 7
# This procedure requires:
#  (1) The "liesymm" package,
#  (2) A closed list of forms of the name R,
#  (3) The the forms be annuled using the "annul" command in "liesymm."
#    (This serves to define the independent and dependent
#    variables for using commands in the "liesymm" package.)
#
#  After the above are satisfied, this procedure can be called.
#  The procedure takes two arguments: (1) A single form from R
#  on which to perform the Lie derivative; (2) A variable name
#  for the multipliers. The procedure performs the Lie derivative
#  and sets it equal the sum of forms in R with multipliers.
#
#  Lie R = q R      (q is a mulitplier, i is fixed, and j sums)
#    V i  j j
#
#  The procedure creates several global variables:
#    (1) mm - a table of equations with multipliers
#    (2) om - a table of equations without multipliers
#    (3) mmnum - number of elements in mm
#    (4) omnum - number of elements in om
#    (5) V - isovector name
#    (6) backmm - second copy of mm
#    (7) backom - second copy of om
#
    local form,nforms,g1,g2,g3,sum,i,h,ES,nterms,\
    ZS,zsnum,remain,mmnum,omnum,origzsnum;
    global mm,om,backmm,backom,con;
    form := F;
    nforms := nops(R);
    con := [cc.(1..nforms)];
    sum := 0;
    for i from 1 to nforms do
      sum := sum + con[i]*R[i]
    od;
    g1 := Lie(form,V) + sum;
    g2 := map(mixpar,map(getcoeff,convert(wcollect(g1),'list')));
    nterms := nops(g2);
```

```
    mmnum := 0;
    omnum := 0;
    ZS := eval(zsname);
    if not type(ZS,'set') then ZS := {} fi;
    origzsnum := nops(ZS);
    ES := [];
    g3 := [];
    for h in g2 do
      g3 := [op(g3),h=0]
    od;
    Dsimp('g3','ZS','hello');
    for h in g3 do
      if has(h,con) then
        mmnum := mmnum + 1;
        mm[mmnum] := h
      else
        omnum := omnum + 1;
        om[omnum] := h;
        ES := [op(ES),h]
      fi
    od;
    backmm := eval(mm);
    backom := eval(om);

    zsnum := nops(ZS) - origzsnum;
    remain := nterms - zsnum - mmnum - omnum;
    esname := ES;
    zsname := ZS;
    print(`THE SET OF TRIVIAL EQUATIONS`);
    print(ZS);
    printf(`\n\n`);
    print(`THE TABLE OF EQUATIONS WITHOUT MULTIPLIERS`);
    for i from 1 to omnum do print(i,om[i]) od;
    printf(`\n\n`);
    print(`THE TABLE OF EQUATIONS WITH MULTIPLIERS`);
    for i from 1 to mmnum do print(i,mm[i]) od;
    printf(`\n\n\n`);
    printf(`%d total equations.\n`,nterms);
    printf(`%d trivial equations.\n`,zsnum);
    printf(`%d equations without multipliers.\n`,omnum);
    printf(`%d equations with multipliers.\n`,mmnum);
    printf(`%d equations arbitrarily eliminated at the whim\n`,remain);
    printf(`   of The Ineffable One.\n`);
  end:



Dsub := proc(x,c,y)
# Version 2
# This procedure does not require liesymm
#
# This procedure helps eliminate multipliers in the global table
#  of equations mm.  The arguments are
#    x -- an equation containing a multiplier.  Only the number is
#         given; the equation is mm[x]
```

```
#    c -- a multiplier to solve for
#    y -- target equations.  can be a list or set, such as [3,5,9]
#         the expression obtained by solving mm[x] for c will be
#         substituted into equations mm[3], mm[5] and mm[9]
#
    local eq1,eq2,con,q,k;
    global mm;
    eq1 := x;
    eq2 := eval(y);
    con := c;
    q := solve(mm[eq1],{con});
    if type(eq2,{'list','set'}) then
      for k in eq2 do
        mm[k] := subs(q,mm[k])
      od
    else
      mm[eq2] := subs(q,mm[eq2])
    fi;
  end:



Dreduce := proc(esname,zsname)
# Version 2
#
#  This routine is called by Dsimp.  Dreduce goes through every
#     equation in zsname, and looks for that term in every equation
#     in esname.  If the term is found, then Dreduce sets it to
#     zero.

  local h,k,func,vars,cc,dd,term,term1,term2,term3,term4,piece1,\
  piece2,num,found,piece,ES,ZS,pos;
  ES := eval(esname);
  ZS := eval(zsname);
  for h in ZS do
    k := [op(op(1,h))];
    func := k[1];
    vars := subsop(1=NULL,k);
    for cc in ES do
      dd := op(1,cc);
      if type(dd,`+`) then term:=[op(dd)]
      else term := [dd]
      fi;
      for term1 in term do
        if type(term1,`*`) then term2 := [op(term1)]
        else term2 := [term1]
        fi;
        for term3 in term2 do
          if type(term3,'function') and has(term3,{'Diff','diff'}) then
            piece := [op(term3)];
            piece1 := piece[1];
            piece2 := subsop(1=NULL,piece);
            num := nops(piece2);
            if piece1 = func then
              found := true;
              for term4 in vars while found do
```

```
              if member(term4,piece2,'pos') then
                piece2 := subsop(pos=NULL,piece2)
              else
                found := false
              fi
            od;
            if found then ES := subs(term3=0,ES) fi
          fi
        fi
      od
    od
  od
od;
zsname := ZS;
esname := ES;
end:
```

```
Dclean := proc(esname,zsname,change)
# Version 3
#
#  This routine is called by Dsimp.  The third argument is a flag.
#  If Dclean makes no changes to esname and zsname, then change
#  is set to "false."  When changes are made then change is set to
#  "true."  This routine searches through esname for any trivial
#  equations.  It looks for single term equations.  It will tear
#  this term apart until it finds a function.  If the term is
#                 -x^2*y*f(x,y) = 0
#  and x,y are independent variables, then f(x,y) = 0.  Dclean
#  eliminates the -x^2*y coefficient, and records that f(x,y)=0,
#  by entering f(x,y) into the set zsname.  Dclean records the number
#  of changes it makes, and reports this information
#
#
  local exlist,added,i,num,deleted,k,cc,xlist,proceed,pp,pp1,\
  ZS,ES,k2,term,numder,numadd,numdel;
  global zlist;
  ZS := eval(zsname);
  ES := eval(esname);
  exlist := NULL;
  added := false;
  deleted := false;
  numadd := 0;
  numdel := 0;
  num := nops(ES);
  if not type(ZS,'set') then ZS := {} fi;
  if not type(zlist,'set') then zlist := {} fi;
  for i from 1 to num do
    k := op(1,ES[i]);
    if type(k,`*`) then
      k2 := [op(k)];
      proceed := true;
    elif type(k,'function') then
      k2 := [k];
      proceed := true
```

```
    else
      proceed := false
    fi;
    if proceed then
      numder := 0;
      for term in k2 do
        if type(term,'function') and has(term,{'Diff','diff'}) then
          pp := term;
          numder := numder + 1
        fi
      od;
      if numder < 2 then
        exlist := exlist,i;
        pp1 := pp = 0;
        zlist := {op(zlist),ES[i]};
        ZS := {op(ZS),pp1};
        added := true;
        numadd := numadd + 1
      fi;
    elif k = 0 then
      exlist := exlist,i;
      deleted := true;
      numdel := numdel + 1
    fi
  od;
  xlist := [exlist];
  change := added;
  if added or deleted then
    cc := nops(xlist);
    for i from cc to 1 by -1 do
      ES := subsop(xlist[i] = NULL,ES)
    od;
    if added then
     printf(`%d equations shifted from ES to ZS\n`,numadd)
    fi;
    if deleted then
     printf(`%d 0 = 0 equations deleted from ES\n`,numdel)
    fi
  fi;
  esname := ES;
  zsname := ZS;
  printf(`   Divide et impera.\n`);
  end:



Dsimp := proc(esname,zsname,mlist)
#
#  This routine serves only to call Dreduce and Dsimp.  The third
#  argument is a flag.  If the third argument is present, then Dsimp
#  assumes that the equations contain multipliers, and calls Dclean2
#  instead of Dclean.  Usually only Dlie calls Dsimp with three
#  arguments.  Dsimp calls Dreduce and Dclean in a loop, and only
#  stops when the logic variable change = "false".  This happens
#  when no further changes can be made.  Dsimp then prints the equations.
#
```

```
  local change,ES,ZS,num;
  ES := eval(esname);
  ZS := eval(zsname);
  ES := map(expand,ES);
  if not type(ZS,'set') then ZS := {} fi;
  change := true;
  if nargs = 3 then
    while change do
      Dreduce('ES','ZS');
      Dclean2('ES','ZS','change')
    od;
  elif nargs = 2 then
    while change do
      Dreduce('ES','ZS');
      Dclean('ES','ZS','change')
    od
  fi;
  ES := map(collect,ES,[wedgeset(0)],'distributed',factor);
  esname := ES;
  zsname := ZS;
  if nargs =2 then
    num := nops(ES);
    print(ES);
    printf(`%d total equations.\n`,num)
  fi;
  end:



Dadd := proc(esname,elist)
# Version 1
#
#  This simple routine adds equations from mm to esname.  Only the
#  numbers from mm are given.  elist can be a single number, or list,
#  or set.
#
  local ES,eqlist,i;
  global mm;
  eqlist := eval(elist);
  ES := eval(esname);
  for i in eqlist do
    ES := [op(ES),mm[i]]
  od;
  esname := ES;
  end:



Dclean2 := proc(esname,zsname,change)
# Version 3
#
#  This is the same as Dclean, only that this procedure additionally
#  searches for any multipliers that equal zero.  Instead of adding
#  the multiplier to zsname, Dclean2 sets the multiplier to zero
#  directly in the equations.  See the notes for Dclean for further
#  information
```

```
#
  local exlist,added,i,num,deleted,k,cc,xlist,proceed,pp,\
  ZS,ES,k2,term,numder,numadd,numdel,func,mult,nummul,muldel;
  global zlist,con;
  ZS := eval(zsname);
  ES := eval(esname);
  exlist := NULL;
  added := false;
  deleted := false;
  muldel := false;
  nummul := 0;
  numadd := 0;
  numdel := 0;
  num := nops(ES);
  if not type(ZS,'set') then ZS := {} fi;
  if not type(zlist,'set') then zlist := {} fi;
  for i from 1 to num do
    k := op(1,ES[i]);
    if type(k,`*`) then
      k2 := [op(k)];
      proceed := true;
    elif type(k,{'function','string'}) then
      k2 := [k];
      proceed := true
    else
      proceed := false
    fi;
    if proceed then
      numder := 0;
      for term in k2 do
        func := type(term,'function') and has(term,{'Diff','diff'});
        mult := type(term,'string') and member(term,con);
        if func then
          pp := term;
          numder := numder + 1;
        elif mult then
          pp := term;
          zlist := {op(zlist),ES[i]};
          ES := subs(pp=0,ES);
          muldel := true;
          exlist := exlist,i;
          nummul := nummul + 1;
          numder := 3
        fi
      od;
      if numder < 2 then
        exlist := exlist,i;
        zlist := {op(zlist),ES[i]};
        ZS := {op(ZS),pp=0};
        added := true;
        numadd := numadd + 1
      fi;
    elif k = 0 then
      exlist := exlist,i;
      deleted := true;
      numdel := numdel + 1
```

```
    fi
  od;
  xlist := [exlist];
  change := added;
  if added or deleted or muldel then
    cc := nops(xlist);
    for i from cc to 1 by -1 do
      ES := subsop(xlist[i] = NULL,ES)
    od;
    if added then
     printf(`%d equations shifted from ES to ZS.\n`,numadd)
    fi;
    if deleted then
     printf(`%d 0 = 0 equations deleted from ES.\n`,numdel)
    fi;
    if muldel then printf(`%d multipliers set to zero.\n`,nummul) fi
  fi;
  esname := ES;
  zsname := ZS;
  printf(`   Divide et impera.\n`);
  end:



Dlie2 := proc(F,cc,esname,zsname)
# Version 1
# This procedure is almost identical to Dlie(). This procedure assumes
# that the Lie derivative has already been performed, and is passed in as
# the first argument.  See the notes to Dlie for further information.
#
    local form,nforms,g1,g2,g3,sum,i,h,ES,nterms,\
    ZS,zsnum,remain,mmnum,omnum,origzsnum;
    global mm,om,backmm,backom,con;
    form := F;
    nforms := nops(R);
    con := [cc.(1..nforms)];
    sum := 0;
    for i from 1 to nforms do
      sum := sum + con[i]*R[i]
    od;
    g1 := form + sum;
    g2 := map(mixpar,map(getcoeff,convert(wcollect(g1),'list')));
    nterms := nops(g2);
    mmnum := 0;
    omnum := 0;
    ZS := eval(zsname);
    if not type(ZS,'set') then ZS := {} fi;
    origzsnum := nops(ZS);
    ES := [];
    g3 := [];
    for h in g2 do
      g3 := [op(g3),h=0]
    od;
    Dsimp('g3','ZS','hello');
    for h in g3 do
      if has(h,con) then
```

```
      mmnum := mmnum + 1;
      mm[mmnum] := h
    else
      omnum := omnum + 1;
      om[omnum] := h;
      ES := [op(ES),h]
    fi
  od;
  backmm := eval(mm);
  backom := eval(om);
  zsname := ZS;
  esname := ES;
  zsnum := nops(ZS) - origzsnum;
  remain := nterms - zsnum - mmnum - omnum;
  print(`THE SET OF TRIVIAL EQUATIONS`);
  print(ZS);
  printf(`\n\n`);
  print(`THE TABLE OF EQUATIONS WITHOUT MULTIPLIERS`);
  for i from 1 to omnum do print(i,om[i]) od;
  printf(`\n\n`);
  print(`THE TABLE OF EQUATIONS WITH MULTIPLIERS`);
  for i from 1 to mmnum do print(i,mm[i]) od;
  printf(`\n\n\n`);
  printf(`%d total equations.\n`,nterms);
  printf(`%d trivial equations.\n`,zsnum);
  printf(`%d equations without multipliers.\n`,omnum);
  printf(`%d equations with multipliers.\n`,mmnum);
  printf(`%d equations arbitrarily eliminated at the whim\n`,remain);
  printf(`   of The Ineffable One.\n`);
end:
```